MEDICAL INVENTORY MANAGEMENT

College Name: BHARATHIDASAN COLLEGE OF ARTS AND SCIENCE

College Code: BRU3D

TEAM ID: NM2025TMID24629

TEAM MEMBERS:

Team Leader Name :

VISHWA S

EMAIL :

Vishwa12050@gmail.com

TEAM MEMBER 1 : SAKTHIYANANTHAN V
EMAIL : Sakthiyananthan8@gmail.com
TEAM MEMBER 2 :SAKTHIKUMAR N
EMAIL : Sakthikumar739711@gmail.com
TEAM MEMBER 3 :VETTAIYAN P
EMAIL : Vettaiyanvettaiyan20@gmail.com

1.INTRODUCTION

1.1 Project Overview

This project is a comprehensive Salesforce application to streamline and manage various operational aspects of medical inventory. The system aims to efficiently maintain supplier details, manage purchase orders, track product details and transactions, and monitor the expiry dates of products. Maintain detailed records of suppliers, including contact information. Catalog product information, including descriptions, stock levels. Monitor and track product expiry dates to avoid using expired items. Comprehensive reports to performance, and purchase orders.

1.2 Purpose

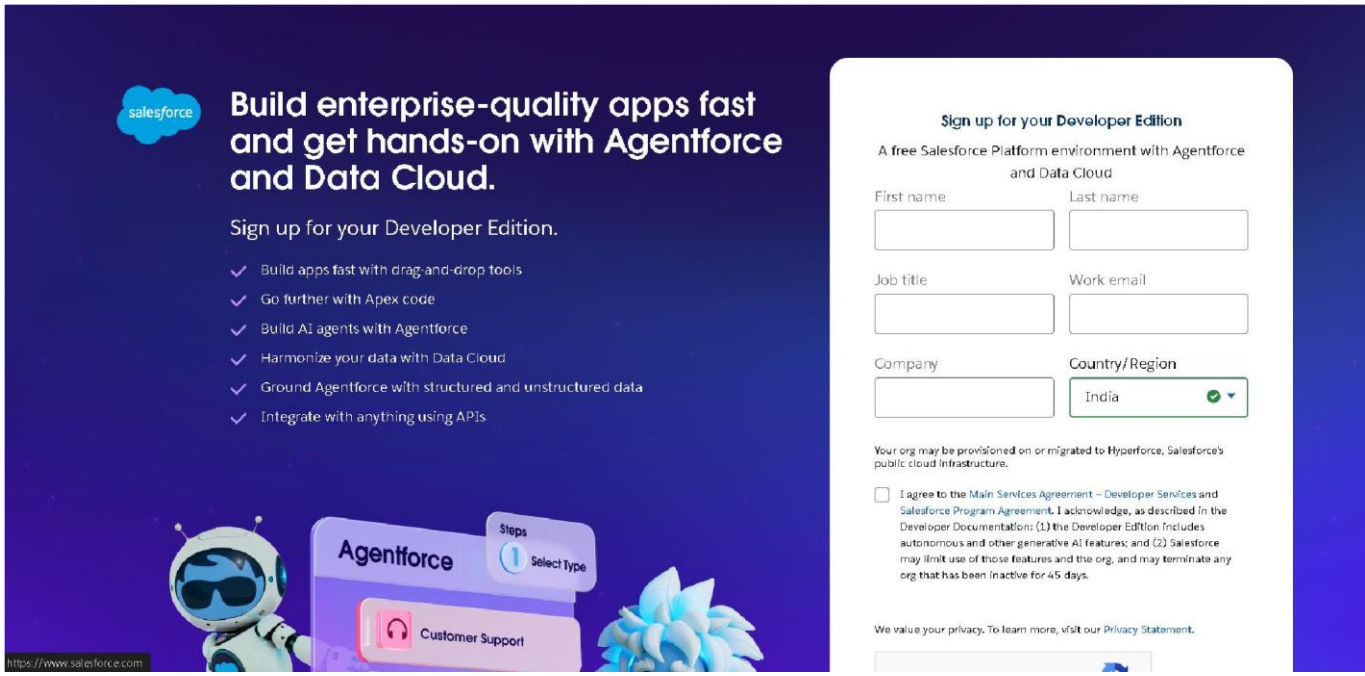
The Medical Inventory Management System is a comprehensive Salesforce application designed to streamline and manage various operational aspects of the medical inventory. It can efficiently maintain supplier details, manage purchase

DEVELOPMENT PHASE

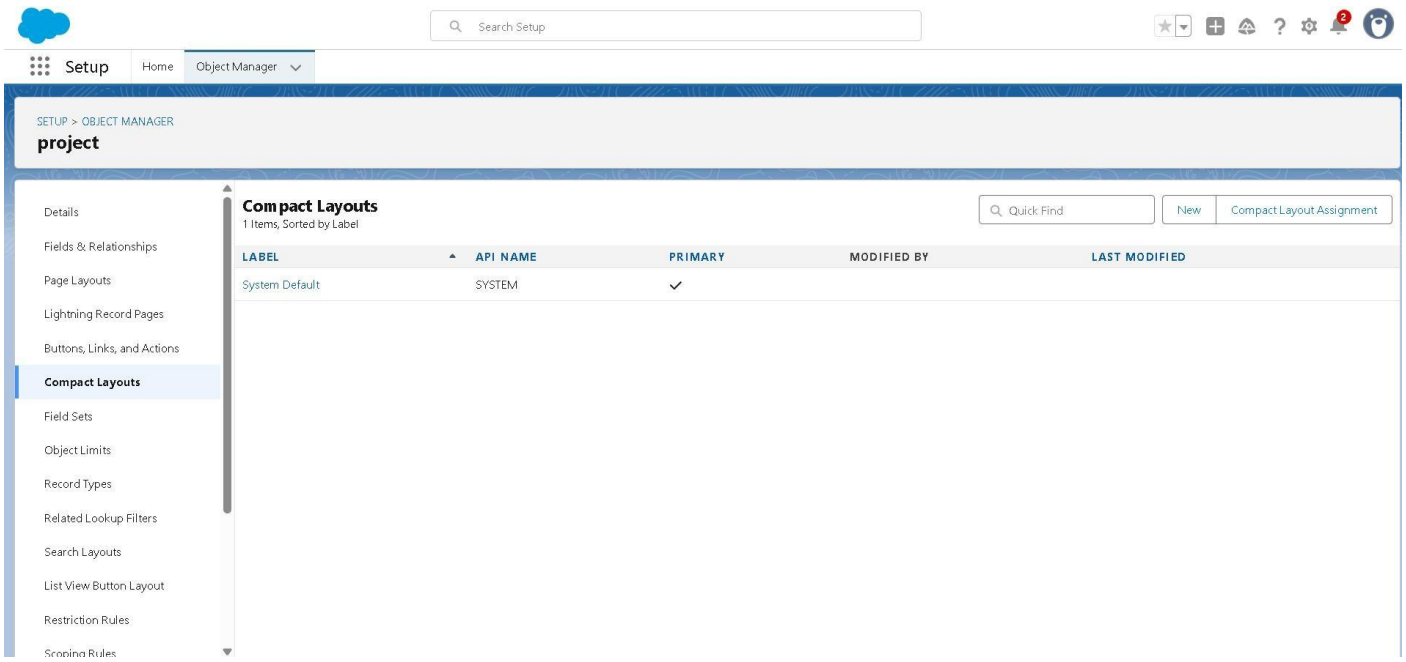
Creating Developer Account:

By using this URL **- https://www.salesforce.com/form/developer-signup/?d=pb**

Salesforce is your customer success platform, designed to help you sell, service, market, analyze, and connect with your customers.



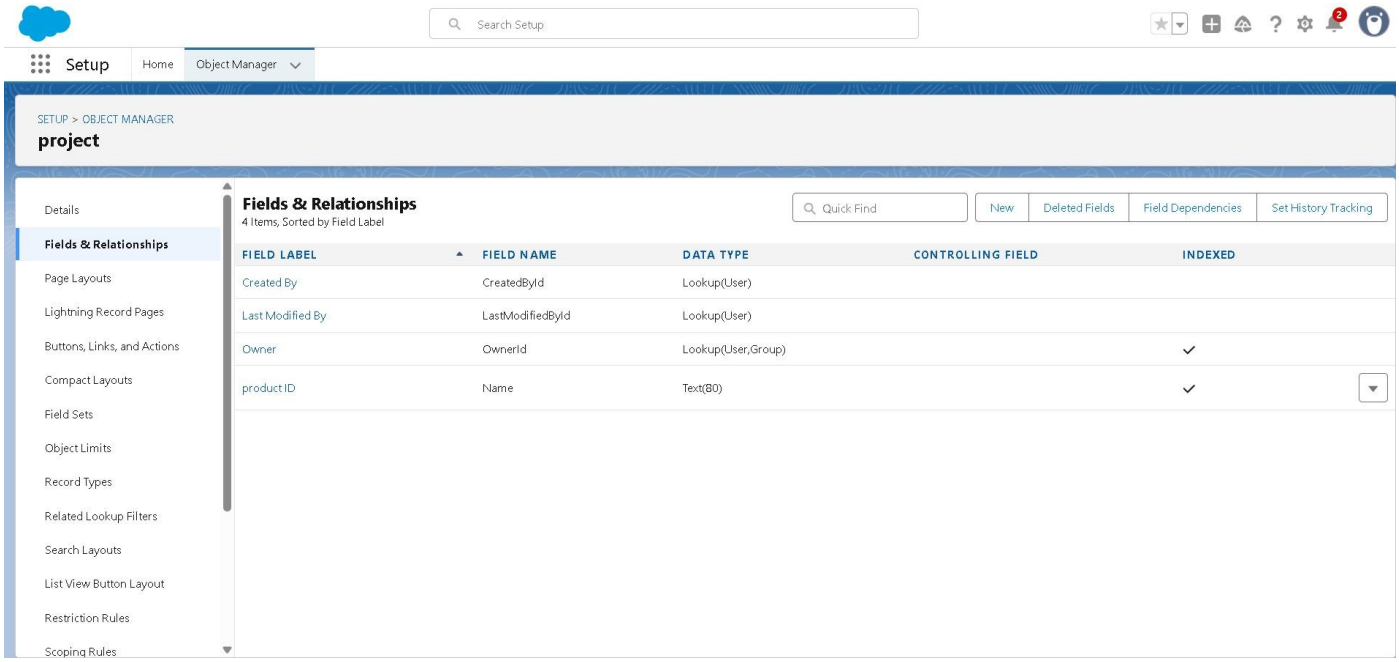
Salesforce has everything you need to run your business from anywhere. Using standard products and features, you can manage relationships with prospects and customers, collaborate and engage with employees and partners, and store your data securely in the cloud. So what does that really mean? Well, before Salesforce, your contacts, emails, follow-up tasks, and prospective deals might have been organized something like this: https://youtu.be/r9EX3lGde5k



A Lightning App in Salesforce is a collection of items that work together to serve a particular function for the end-users. These items can include standard and custom objects, tabs, utilities, and other productivity tools. Lightning Apps are designed to provide a more intuitive and efficient user experience compared to traditional Salesforce apps

>>COMPACT LAYOUT

>FIELDS & RELATIONSHIP



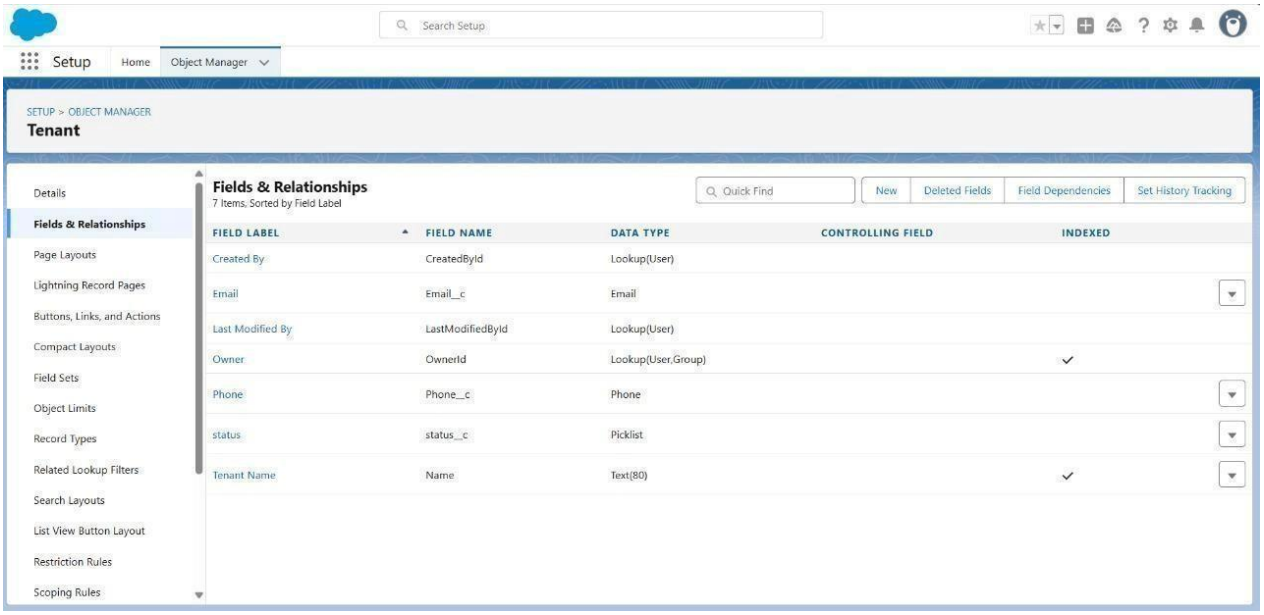
Object	Field Name	Data Type
Product	Product ID(Standard)	Text
	Product Name	Text
	Product Description	Text Area
	Minimum Stock Level	Number(18, 0)
	Current Stock Level	Number(18, 0)
	Unit Price	Currency(16, 2)
	Expiry Date	Date
Purchase Order	Purchase Order ID(Standard)	Text
	Supplier ID	Lookup(Supplier)
	Order Date	Date
	Expected Delivery Date	Date
	Actual Delivery Date	Date
	Order Count	Roll-Up Summary (COUNT Order Item)
	Total Order Cost	Currency(16, 2)
Order Item	Order Item ID(Standard)	Text
	Product ID	Lookup(Product)
	Purchase Order ID	Master-Detail(Purchase Order)
	Quantity Ordered	Number(18, 0)
	Quantity Received	Number(18, 0)
	Unit Price	Formula(Currency)
	Amount	Formula(Currency)

Inventory Transaction	Transaction ID(Standard)	Text
	Purchase Order ID	Lookup(Purchase Order)
Supplier	Transaction Date	Date
	Transaction Type	Picklist
	Total Order Cost	Formula(Currency)
	Supplier ID(Standard)	Text
	Supplier Name	Text
	Contact Person	Text
	Phone Number	Phone
	Email	Email
	Address	TextArea

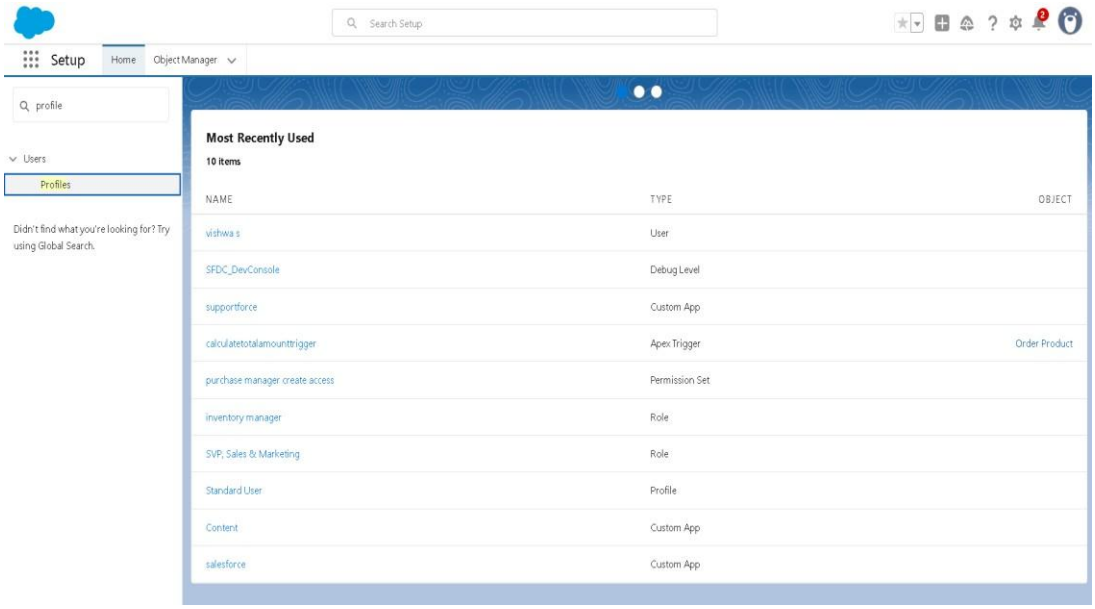
Activity 1: Creating a Text Field in Product Object

To create fields in an object:

1. Click the gear icon and select Setup. This launches Setup in a new tab.
2. Click the Object Manager tab next to Home.
3. Select Product custom object.
4. Select Fields & Relationships from the left navigation
5. Click on New
6. Select Text field, click Next
7. Enter Field Label as “Product Name” and Length 255.
8. Select Required Field.
9. Click Next, Next, then Save & New.



- Developed Profile App with relevant tabs



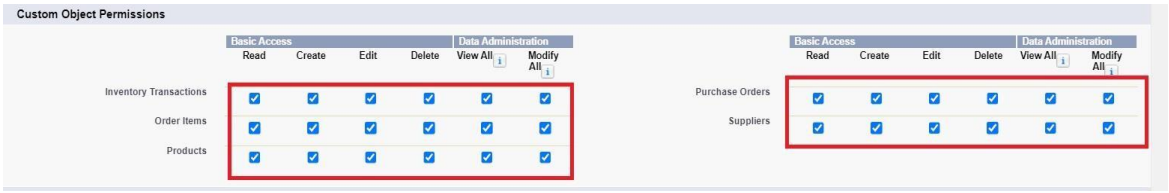
Activity 1: To create an Inventory Manager Profile

1. Go to setup >> type profiles in quick find box >> click on profiles >> clone the desired profile (Standard User) >> enter profile name (Inventory Manager) >> Save.

   While still on the profile page, then click Edit.

2. Select the Custom App settings as default for the Medical Inventory Management.

3. Scroll down to Custom Object Permissions and Give access permissions as mentioned in the below diagram.

4. 

5.

6. Change the password policies as mentioned :

7. User passwords expire in should be “ never expires ”.

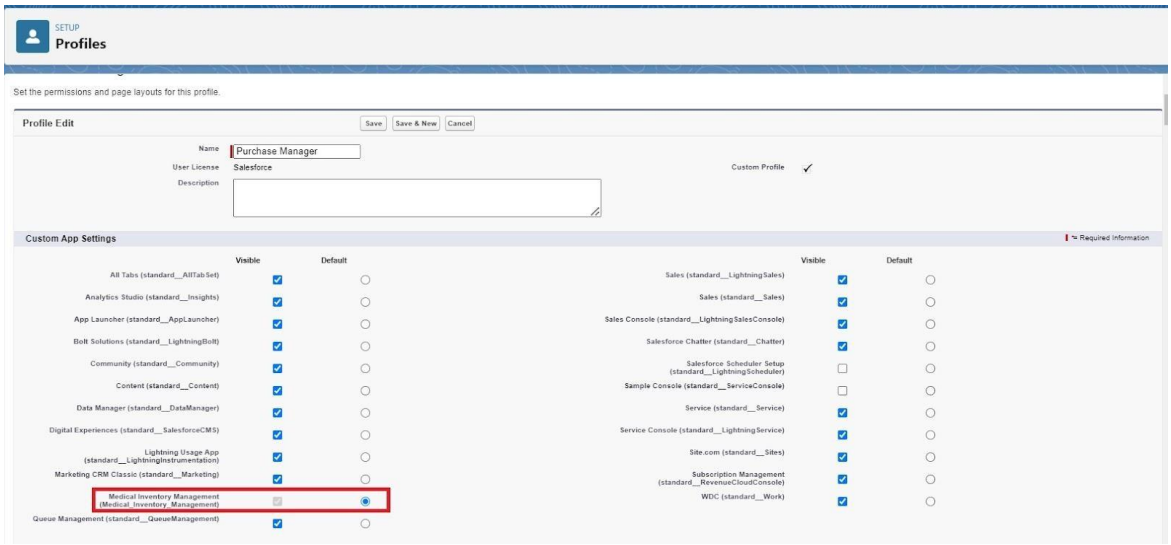
8. Minimum password length should be “ 8 ”, and click save.

Activity 2: To create an Purchase Manager Profile

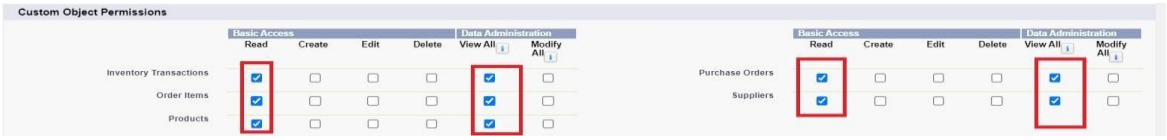
1. Go to setup >> type profiles in quick find box >> click on profiles >> clone the desired profile (Standard User) >> enter profile name (Purchase Manager) >> Save.

2. While still on the profile page, then click Edit.

3. Select the Custom App settings as default for the Medical Inventory Management.



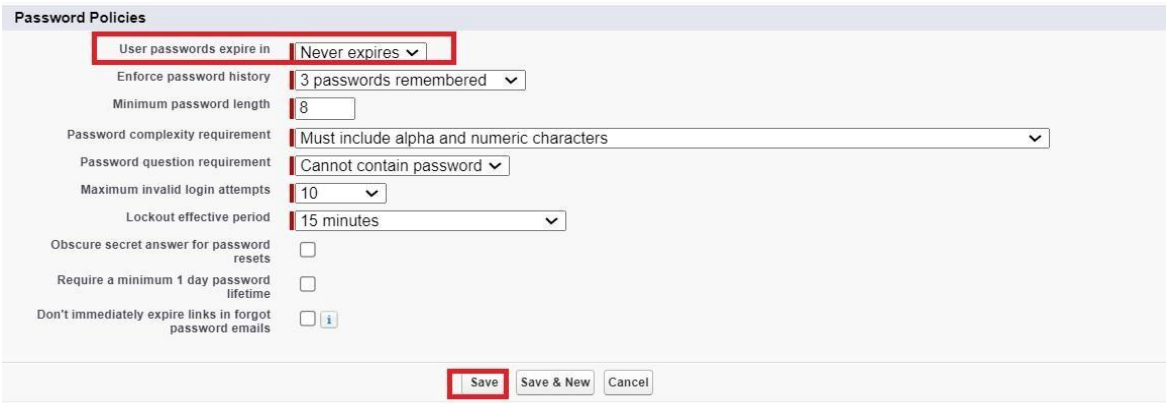
4. Scroll down to Custom Object Permissions and Give access permissions as mentioned in the below diagram.



5. Change the password policies as mentioned :

6. User passwords expire in should be “ never expires ”.

7. Minimum password length should be “ 8 ”, and click save.



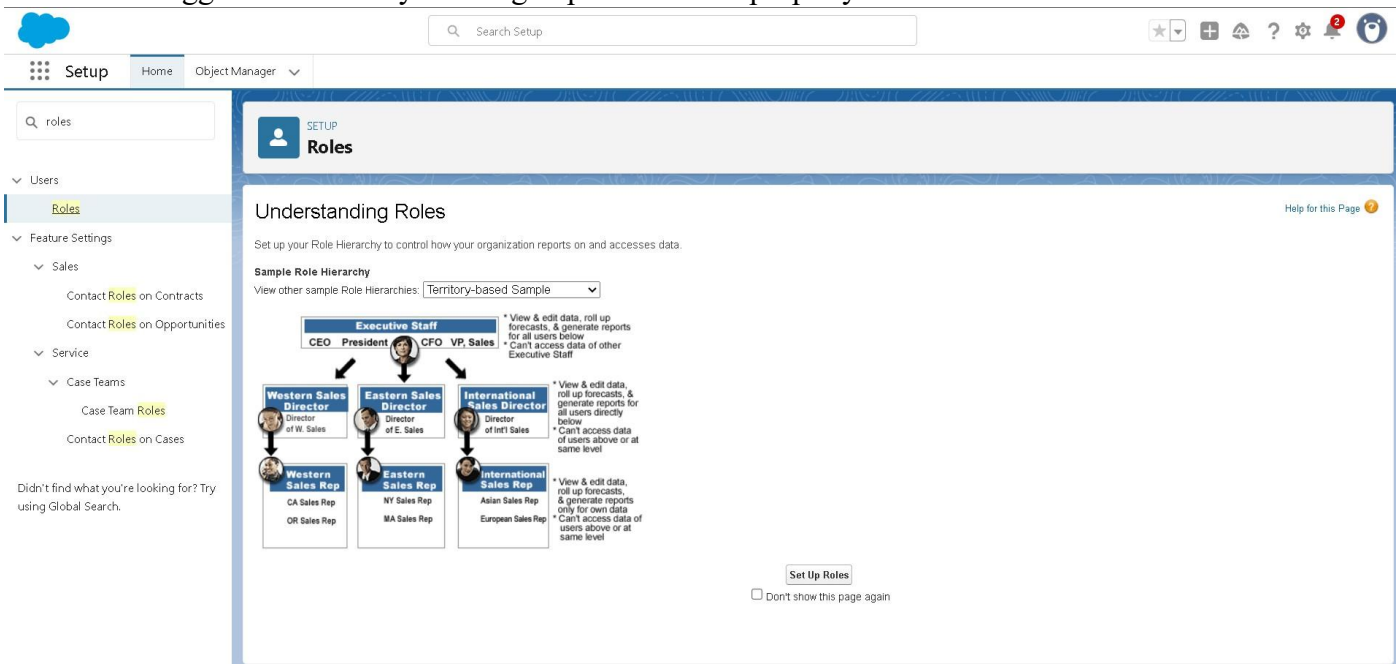
- implemented Flows for monthly rent and payment success

- To create a validation rule to a Lease Object

FUNCTIONAL AND PERFORMANCE TESTING

Performance Testing

- Trigger validation by entering duplicate tenant-property records



- Validation Rule checking

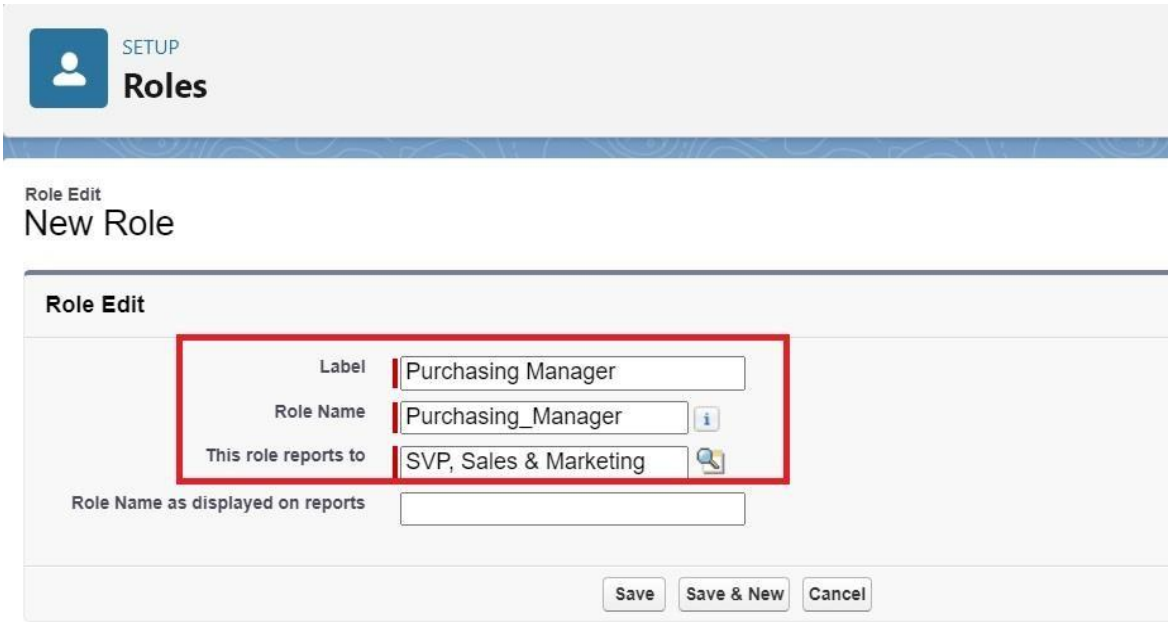
Roles in Salesforce are used to control record-level access and define the hierarchy of an organization, determining the level of visibility and sharing of records among users. Roles work in conjunction with profiles to provide a robust security model. While profiles control what actions users can perform (object and field permissions), roles control which records users can see based on their position in the hierarchy.

Activity 1 : Create a Purchasing Manager Role.

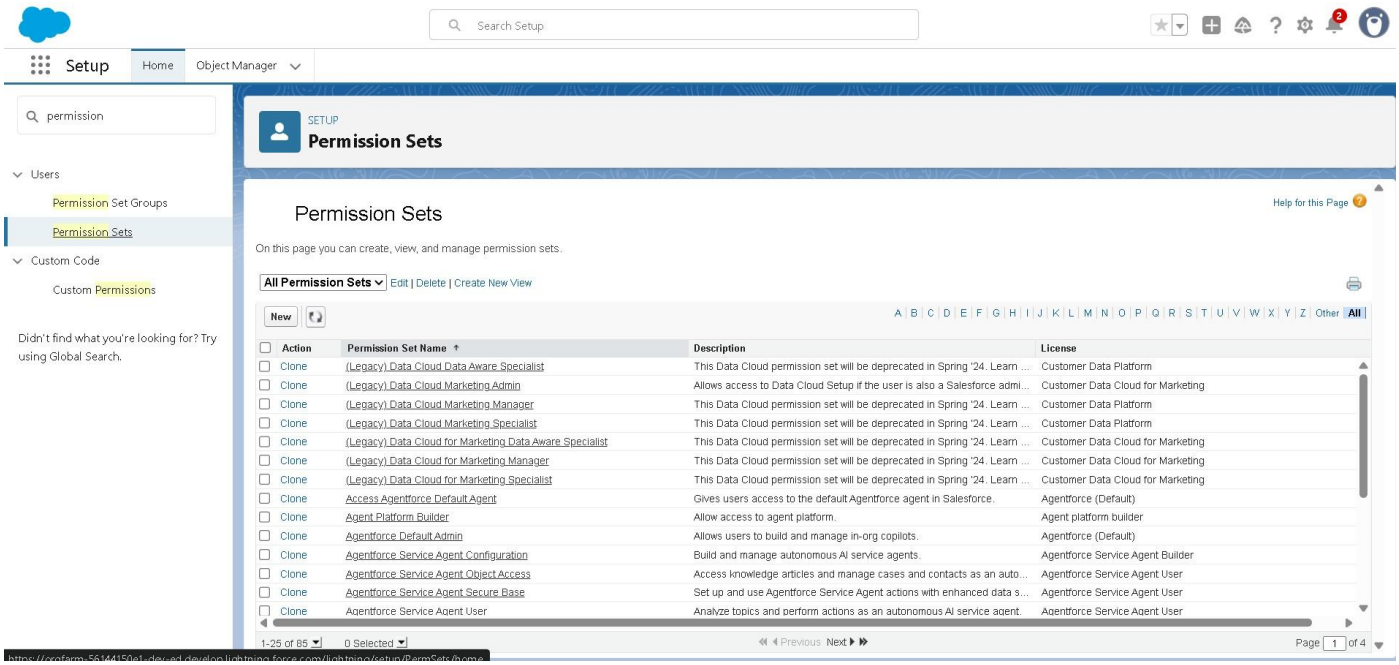
1. Go to quick find >> Search for Roles >> click on Set Up Roles.

2. Click on Expand All and click on add role under SVP, Sales & Marketing role.

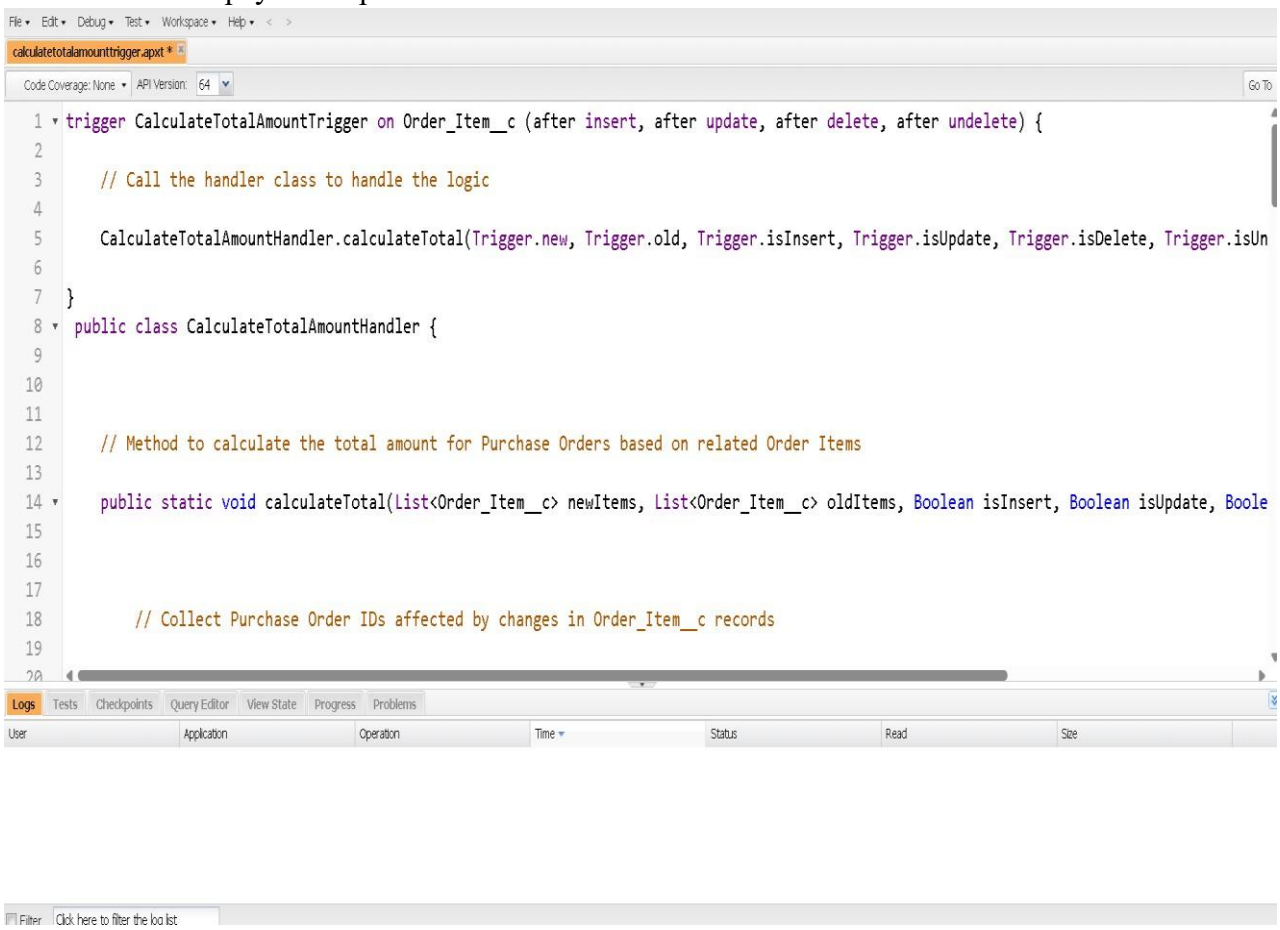
3. Give Label as "Purchasing Manager" and Role name gets auto populated. Then click on Save.



PERMISSION SETS

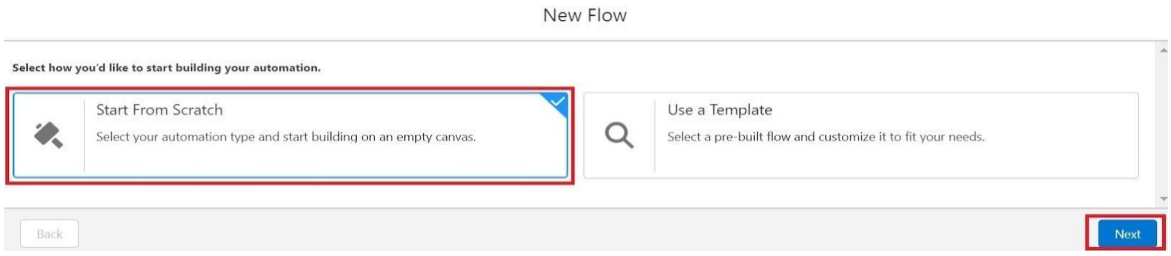


- Test flows on payment update

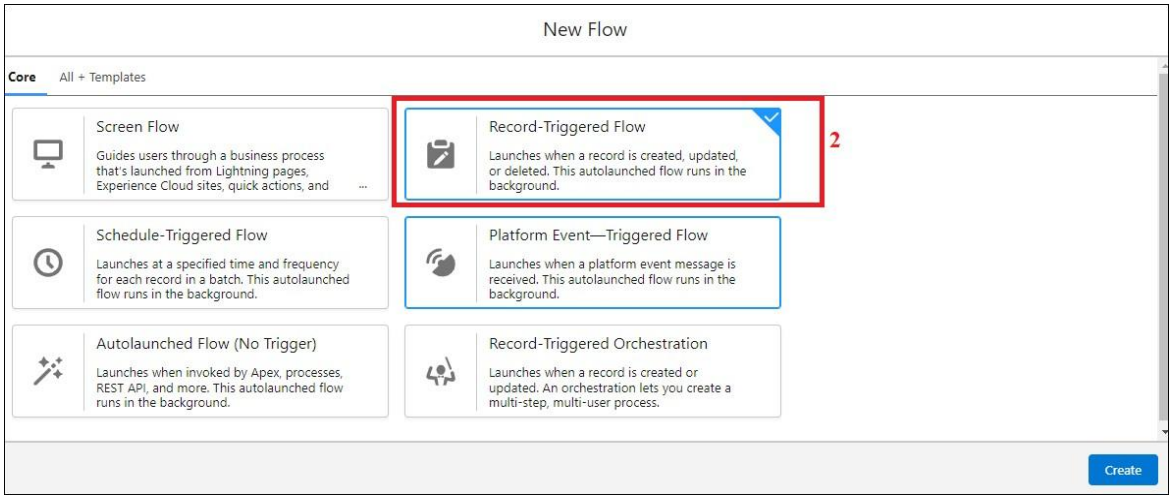


- Approval process validated through email alerts and status updates
- **Activity 1 : Create Flow to update the Actual Delivery Date.**

2. Go to setup >> type Flow in quick find box >> Click on the Flow and Select the New Flow >> Start From Scratch .

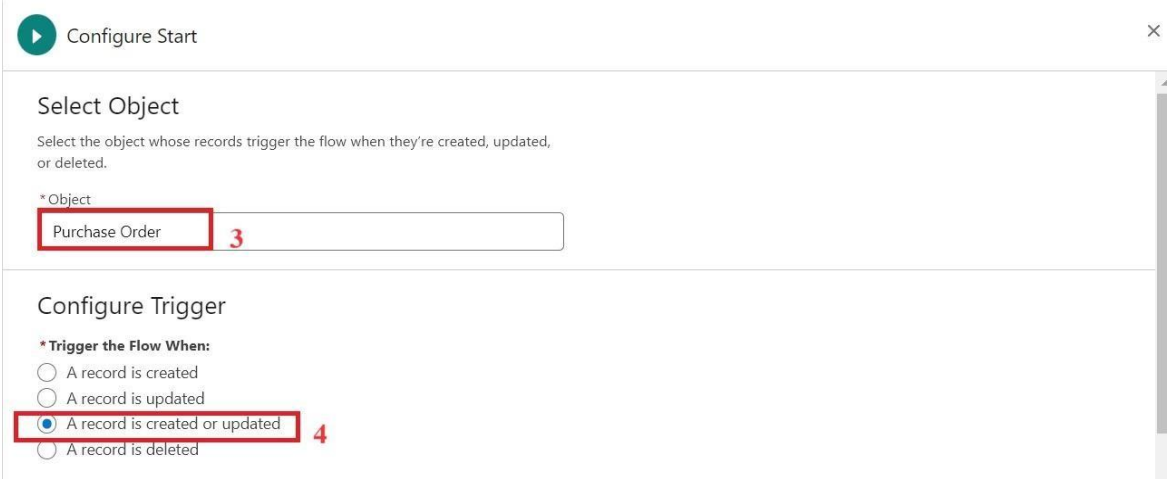
- 

3. Select the record Triggered flow.Click on create.

- 

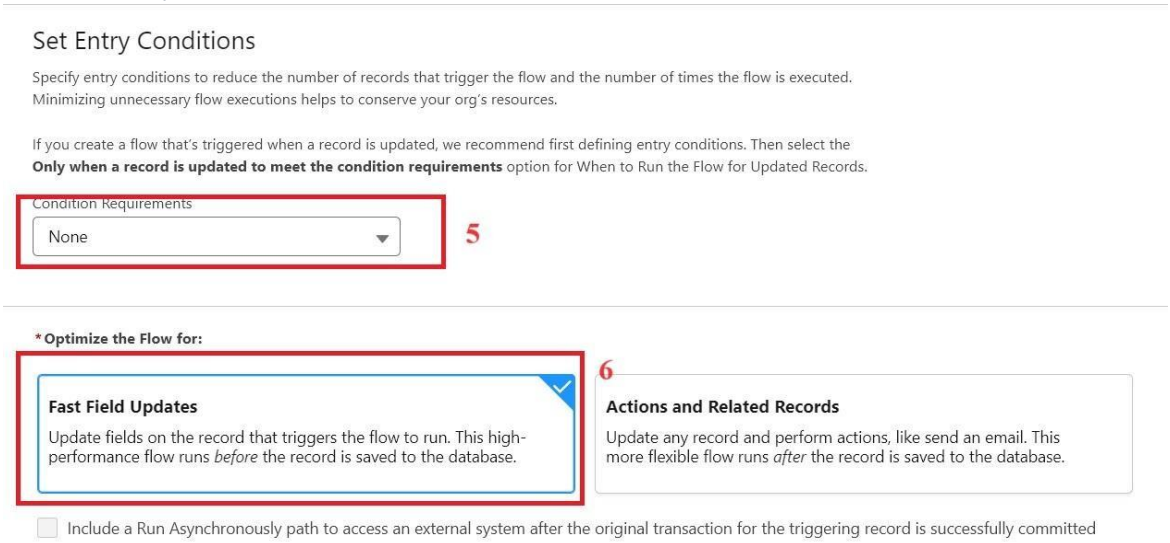
4. Under Object select "Purchase Order"

5. Select A record is created or updated

- 

6. Set Entry Conditions : None

7. Select Fast Field Updates and click on Done

- 

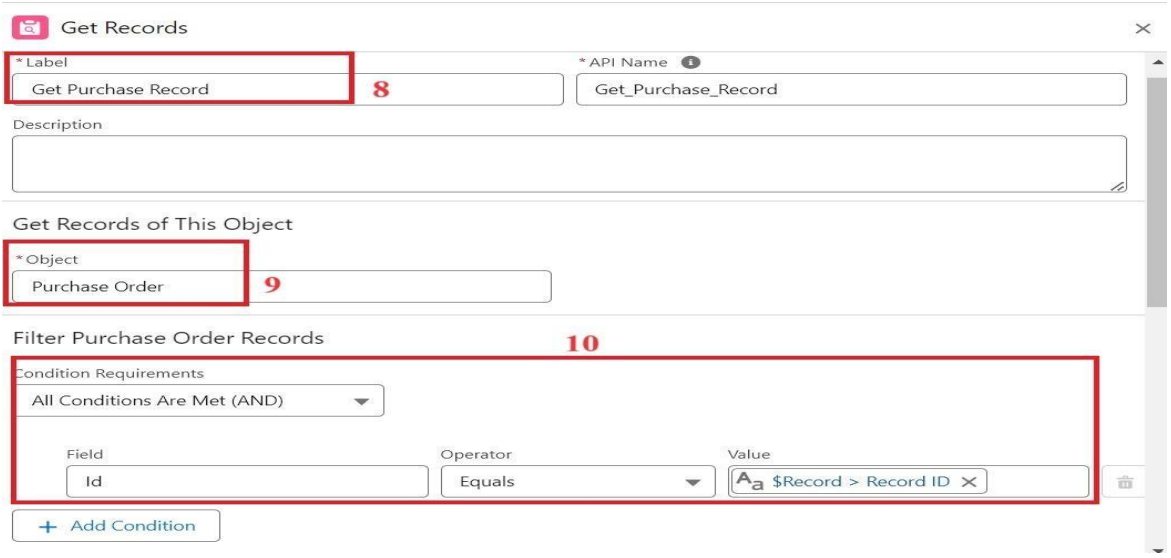
8. Under the record trigger flow click on the "+" icon and select Get Records.

9. Enter Label as " Get Purchase Record ".

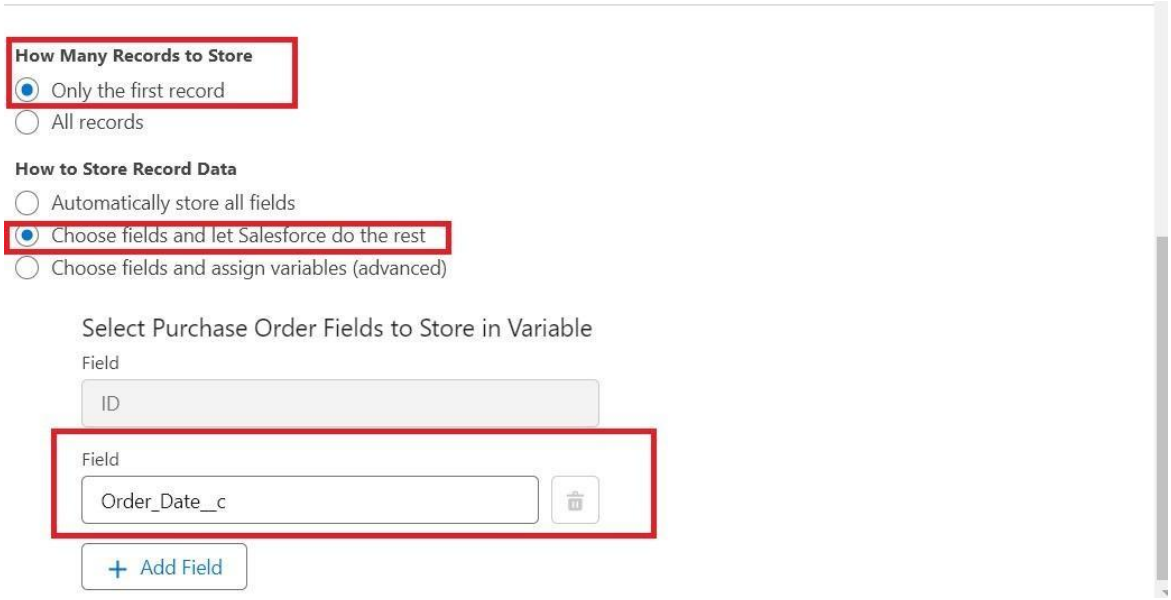
10. For Object select Purchase Order.

11. For Condition Requirements , select All Conditions are Met(AND)

- For the first condition select as follows:
- Field: Id
- Operator: Equals
- Value: {!$Record.Id}

- 

12. For How many Records to store Select Only the First Record. **13.** For How to Store Record Data select Choose fields and let Salesforce do the rest. Select Field: Order_Date__c. Click on Done.

- 

14. In the Flow Builder, click on the Manager tab on the left-hand side >> Click on New Resource >> In the Resource Type dropdown, select Variable.

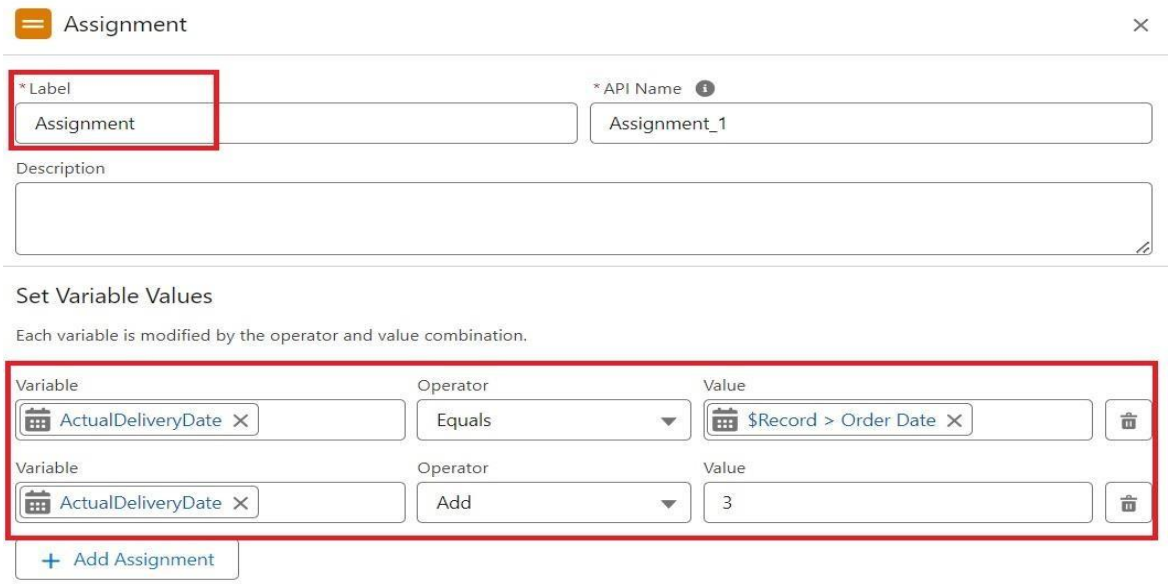
15. Enter API name as ActualDeliveryDate >> Select Data type as Date >> Click on Done.

16. From the Toolbox drag and drop Assignment element.

17. Enter the label as “Assignment”.

18. Set Variable Values:

- a) Variable : {!ActualDeliveryDate}
- Operator : Equals
- Value : {!$Record.Order_Date__c}
- b) Variable : {!ActualDeliveryDate}
- Operator : Add ● Value : 3
-
-



-

19. Click Done

20. From the Toolbox drag and drop Update Records element and connect to the Assignment element.

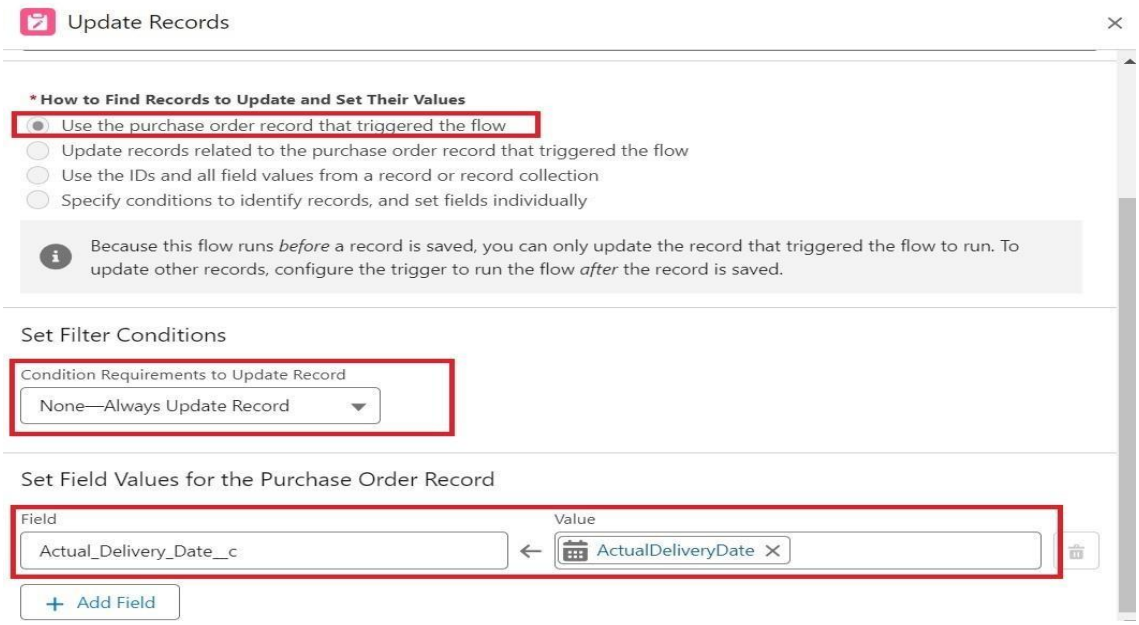
21. Enter the label as “Updating Purchasing Order”.

22. How to Find Records to Update and Set Their Values : Use the Purchase Order record that triggered the flow

23. Set Filter Conditions : None -Always Update Record

24. Set Field Values for the Trip Record as

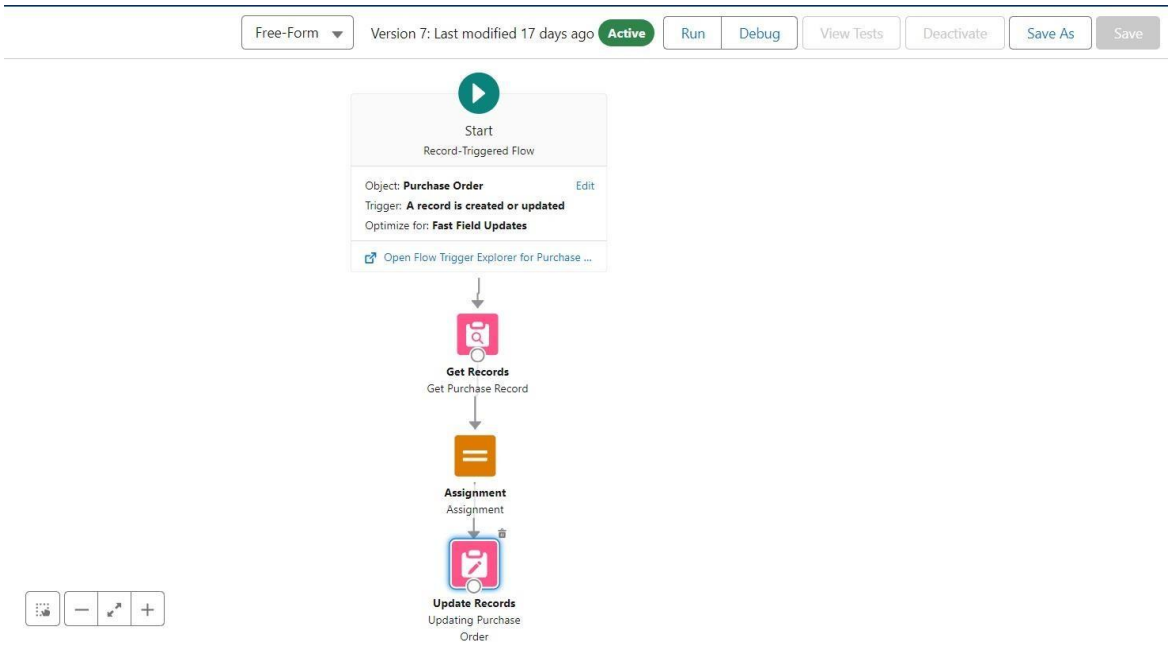
- Field : Actual_Delivery_Date__c
- Value : {!ActualDeliveryDate}



25. Click Done

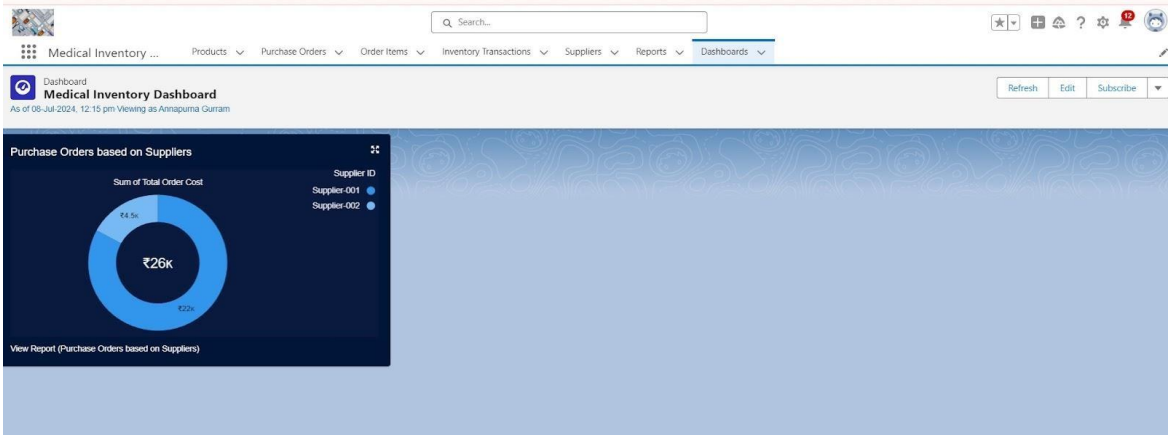
26. Save the flow as “Actual Delivery Date Updating”.

27. Activate the flow.



RESULTS

Output Screenshots



- Tabs for Property, Tenant, Lease, Payment
- Email alerts
- Request for approve the leave
- Leave approved
- Leave rejected

- Flow runs

- Trigger error messages

ADVANTAGES & DISADVANTAGES

.

CONCLUSION

The Lease Management System successfully streamlines the operations of leasing through a structured, automated Salesforce application. It improves efficiency, communication, and data accuracy for both admins and tenants.

APPENDIX

- **Source Code:** Provided in Apex Classes and Triggers **Test.apxt:** trigger CalculateTotalAmountTrigger on Order_Item__c (after insert, after update, after delete, after undelete) {

```
   // Call the handler class to handle the logic
   CalculateTotalAmountHandler.calculateTotal(Trigger.new, Trigger.old, Trigger.isInsert, Trigger.isUpdate, Trigger.isDelete, Trigger.isUndelete);
}
```

Step 4:

i) In the Developer Console window, go to the top menu and click on "File".

ii)Select New: From the dropdown menu under "File", select "New".

iii)Choose Apex Class: Name it as CalculateTotalAmountHandler

```
 public class CalculateTotalAmountHandler {

   // Method to calculate the total amount for Purchase Orders based on related Order Items     public static void calculateTotal(List<Order_Item__c> newItems, List<Order_Item__c> oldItems, Boolean isInsert, Boolean isUpdate, Boolean isDelete, Boolean isUndelete) {

      // Collect Purchase Order IDs affected by changes in Order_Item__c records
      Set<Id> parentIds = new Set<Id>();

      // For insert, update, and undelete scenarios        if (isInsert || isUpdate || isUndelete) {           for (Order_Item__c ordItem : newItems) {
parentIds.add(ordItem.Purchase_Order_Id__c);
         }
      }

      // For update and delete scenarios        if (isUpdate || isDelete) {
         for (Order_Item__c ordItem : oldItems) {              parentIds.add(ordItem.Purchase_Order_Id__c);
         }
      }

      // Calculate the total amounts for affected Purchase Orders
      Map<Id, Decimal> purchaseToUpdateMap = new Map<Id, Decimal>();
```

```
    if (!parentIds.isEmpty()) {
        // Perform an aggregate query to sum the Amount__c for each Purchase Order
        List<AggregateResult> aggrList = [
            SELECT Purchase_Order_Id__c, SUM(Amount__c) totalAmount
            FROM Order_Item__c
            WHERE Purchase_Order_Id__c IN :parentIds
            GROUP BY Purchase_Order_Id__c
        ];

        // Map the result to Purchase Order IDs            for
(AggregateResult aggr : aggrList) {
            Id purchaseOrderId = (Id)aggr.get('Purchase_Order_Id__c');            Decimal
totalAmount                    =                    (Decimal)aggr.get('totalAmount');
purchaseToUpdateMap.put(purchaseOrderId, totalAmount);
        }

        // Prepare Purchase Order records for update
        List<Purchase_Order__c> purchaseToUpdate = new List<Purchase_Order__c>();            for (Id
purchaseOrderId : purchaseToUpdateMap.keySet()) {
            Purchase_Order__c purchaseOrder = new Purchase_Order__c(Id = purchaseOrderId,
Total_Order_cost__c = purchaseToUpdateMap.get(purchaseOrderId));            purchaseToUpdate.add(purchaseOrder);
        }

        // Update Purchase Orders if there are any changes            if
(!purchaseToUpdate.isEmpty()) {
            update purchaseToUpdate;
        }
    }
  }
}
```

Save it.